BiFS based approaches to remote display for mobile thin clients

M. Mitrea[*1] P. Simoens[2] B. Joveski[1] J. Marshall[3] A. Taguengayte[3] F. Prêteux[1] B. Dhoed[2]

[1] Institut TELECOM / TELECOM SudParis /ARTEMIS Department, France
[2] IBBT & Ghent University, Belgium
[3] Prologue Software, France

ABSTRACT

Under the framework of the FP-7 European MobiThin project, the present study addresses the issue of remote display representation for mobile thin client. The main issue is to design a compressing algorithm for heterogeneous content (text, graphics, image and video) with low-complex decoding. As a first step in this direction, we propose a novel software architecture, based on BiFS – Binary Format for Scenes (MPEG-4 Part 11). On the server side, the graphical content is parsed, converted and binary encoded into the BiFS format. This content is then streamed to the terminal, where it is played on a simple MPEG player. The viability of this solution is validated by comparing it to the most intensively used wired solutions, *e.g.* VNC - Virtual Network Computing.

Key words: mobile thin client, remote display, BiFS, MPEG-4 Part 11.

1. INTRODUCTION

Thin client solutions (*i.e.* processing activities depending primarily on the server) have been successful in wired LAN settings: cost reductions, inherent data security and privacy, more efficient use of resources, ubiquitous data and service access are their main advantages. Despite the success in the wired scenario, solutions able to perform equally well in a wireless wide area network still do not exist because of network and device heterogeneity. The main deadlocks in this respect are taken into account by the European ICT MobiThin project[1]. They include architecture and technology issues (wireless medium optimization, dedicated video codec and user pattern research, software/middleware, performance and energy saving oriented solutions), as well as economic ones (business models and structures). All these different aspects are to be accommodated into an integrated prototype for the wireless thin client.

Under this framework, the present contribution addresses the issue of the remote display, Figure 1. The remote display is the client end-point responsible for making visible graphical application output received from the server. The server is in charge of graphical content computation: it detects active applications, collects their graphical output and converts it into binary compressed format. This compressed graphical content is intended for display, with minimal hardware requirements, on the client side. The communication between the server and the client is ensured by a solution dependent, both hardware and software, bi-directional protocol: while the downlink is devoted to the graphical content, the uplink is mainly used for client events (typically, user interaction such as keystrokes and pointing device actions).

In order to achieve this task for conventional wired thin client environments, several sound technical solutions emerged on the market, like Virtual Network Computing (VNC[2]), Citrix' Independent Computing Architecture[3] or Microsoft's Remote Desktop (RDP[4]). However, a lot of improvements are still required for efficient mobile thin clients. Firstly, the network

[*] Corresponding author: Dr. Mihai Mitrea, Telecom SudParis, 9, rue Charles Fourier, 91011 Evry, France
mihai.mitrea@it-sudparis.eu, www.it-sudparis.eu/artemis, www.mobithin.eu

constraints are far more restrictive, both in terms of bandwidth and reliability. Secondly, the mobile terminal is significantly less complex equally in terms of software as of hardware resources (computing power, graphical facilities, and power autonomy).

The MobiThin consortium approaches this issue by considering the MPEG-4 based solutions BiFS[5,6] and LASeR[7]. The primary reason behind this choice being their ability to efficiently combine, represent and compress heterogeneous content, thus alleviating bandwidth constraints. Secondly, the possibility of installing low complexity players on virtually any mobile thin client more than comforts our choice in this respect.

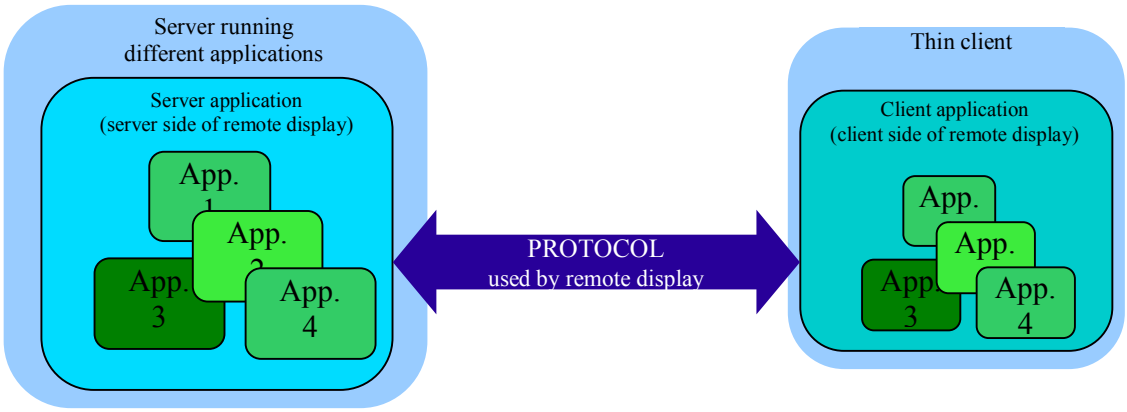


Fig. 1. Remote viewer synopsis.

The present paper intends to be a proof of concept (PoC) for BiFS-based remote display solutions. Note, that any alternative solution for a mobile client remote display, should also abide by the general constraints of genericity (*i.e.* the possibility of deployment irrespective of the thin terminal particularities), low complexity, and backward compatibility.

This paper has the following structure. After this introduction in Section 1, Section 2 outlines the principles of BiFS. Section 3 proposes an architecture using BiFS as the core component for remote application display. Section 4 describes the experimental setup. Section 5 presents the results obtained for the visual quality of the displayed content, the compression efficiency and the viability of event management. Section 6 concludes the paper discussing the perspectives for our future work.

2. BIFS AT A GLANCE

BiFS is an MPEG-4 scene description language designed for the optimization of interactive rich-media services (text, audio, video, 2D & 3D graphics) and this at the representation, delivery and rendering levels. Furthermore, in addition to its ancestors (*e.g.* VRML - Virtual Reality Modeling Language), BiFS provides distinguishing mechanisms such as scene updates, binary compression and data streaming.

Content representation

In order to support all of these features, BiFS exhibits an object-oriented and a stream-based design. All content is described by a scene-graph, providing a hierarchical representation of audio, video and graphical components. Consequently, each object becomes a BiFS node with abstract interfaces thus affording for the manipulation of its properties, independent of its media. Each node contains objects, either individual or grouped, to be displayed, transformations specifying the spatial coordinates of the objects, and a list of fields defining the particular behavior of the considered node. For example, a Box

node has width, height and depth fields specifying the size of the box. MPEG-4 has around 100 nodes with 20 basic field types: boolean, integer, floating point, two- and three-dimensional vectors, time, normal vectors, rotations, colors, URLs, strings, images, and other more arcane data types such as scripts.

BiFS compression

The scene-graph structure allows high level description of the graphical content and makes the compression independent of the source media (video, images, audio, *etc*…): the coding is only performed on the scene-graph description, the underlying media encoding remaining unchanged.

User interaction

BiFS provides a well-defined framework for interaction between elements and for dealing with user-input (*e.g.* mouse click). Technically, user input is also represented as a BIFS node (*e.g.* TouchSensor and InputSensor) extending the functionalities of content nodes to be able to express complex behaviors (*e.g.* video play/stop). Additional interactivity can be provided by translating application events into local BiFS scene description updates. More sophisticated interactive applications can be created by using the programming features of MPEG-4 Systems (JAVAScript).

3. BIFS-BASED ARHITECTURE FOR REMOTE DISPLAY

The implementation of a BiFS-based remote display exploits the MPEG-4 capabilities of (1) representing hybrid graphical content and (2) efficiently compressing video/image/graphics. The MobiThin solution depicted in Figure 2 bridges the gap between this more general idea and a PoC.

Any remote display solution must provide for the needs of server components (where the graphical content is processed), client components (where the graphical content is displayed) and the protocol governing communication between the client and the server.

In the traditional (legacy) X Window System, the graphical content is produced by the XServer in response to XClient requests. Communication between the XClient and the XServer is described by the XProtocol. The MobiThin solution imposes no modifications what so ever at the application level; instead, it simply takes as input the graphical content conveyed by the XProtocol. Actually, the architecture advanced in Fig. 2 comprises the following components:

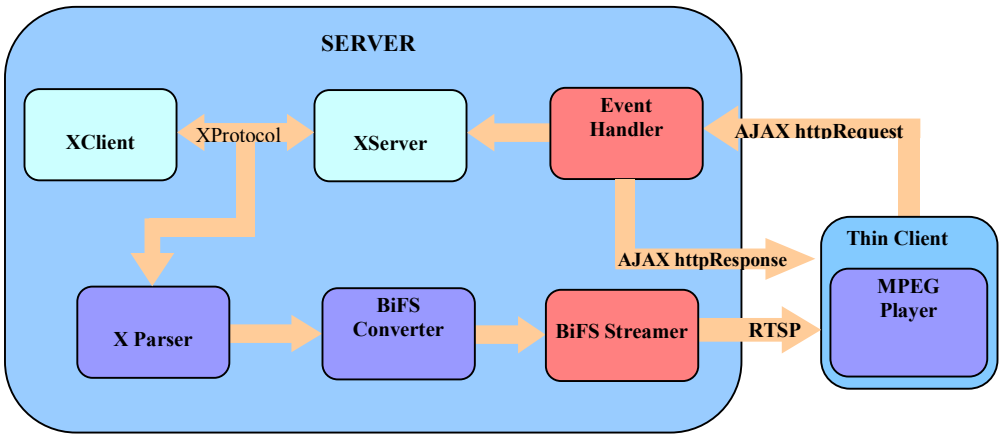


Fig. 2. BiFS-based architecture for remote graphical display.

On the server side:

- the Parser intercepts the graphical content from the XProtocol;
- the Converter translates it into a BiFS scene graph and then performs binary compression;
- the Streamer ensures the transmission of the BiFS content from the server to the thin client using RTSP;
- the Event Handler manages user interaction, intercepting and redistributing the user activity events (represented by AJAX HTTP Requests[8]).

On the client side:

- the player displays the BiFS content and captures the user-interaction.

4. EXPERIMENTAL SETUP

In order to implement the architecture in Figure 2, two virtual machines (VMs) operating on a host PC (running the Fedora 9) are set-up. The first VM is a Linux Ubuntu version and acts as the server in the architecture, thus accommodating the XClient, the XParser, the BiFS Converter, the BiFS Streamer, and the Event handler. The second VM is a basic Linux version (the XORG installation, without any graphical interface) and acts as the XServer. The Thin client is emulated as an MPEG-4 player running on the host PC (outside of VMs).

The XClient, the XServer & the XProtocol

These X components are part of the X Window System providing a graphical user interface (GUI) and generating all the needed graphical primitives for further processing.

The XParser

This component was developed for the parsing of the XProtocol between the XClient and XServer in order to extract the graphical primitives to be rendered. Neither XServer nor XClient are aware of this parser. By listening on a socket through which the XServer and XClient communicate, the parser intercepts the XProtocol messages, performs parsing and passes the results to the BiFS converter.

The BiFS Converter

Every single X request intercepted by the parser is mapped to a function which converts it to its BiFS counterpart. After the conversion, binary encoding is performed using the BiFS encoder function provided by the GPAC Framework[9]. Amongst the X graphical requests already converted, we may mention: CreateGC, PolyFillRectangle, PolyRectangle, PolySegment, FillPoly, PolyLine, CopyArea, PolyText8, CreateWindow, ConfigureWindow, PutImage, CreatePixmap, *etc*... Note that these 12 graphical primitives are the most important and most frequently used.

The MPEG player

Under the project architecture, the GPAC player, as part of GPAC multimedia solution packet, has been considered and retained. It is supported on Windows platforms (PC or PocketPC) and all platforms with GCC, SDL 1.2. The GPAC player supports rendering of 2D/3D and mixing 2D/3D drawings (documents mixing BiFS, SVG, VRML/X3D, LASeR).

The player is also in charge of capturing user interaction and of its interpretation into AJAX HTTP Requests.

The Event handling

This was developed for the reception of AJAX HTTP Requests, and according to the request nature, it can provide two different types of processing. On the one hand, the *Event handling* may convert the AJAX HTTP Requests and post them to

the XServer for further processing. The XServer would then ensure the XServer event management, *i.e.* would update the scene, through the down link, with the new graphical information for the scene (XServer updates). On the other hand, the scene updates may be sent back to the player directly as AJAX HTTP Responses.

The BiFS streamer

This component is implemented as an independent application. It integrates the streaming support from the LIVE555 Streaming Media[10] and the GPAC libraries for the handling of BiFS graphical content. The input to the streamer is BiFS MPEG-4 content with its output being sent to the thin client over an RTSP protocol. At the present moment, the streamer is only capable of streaming BiFS MPEG-4 files.

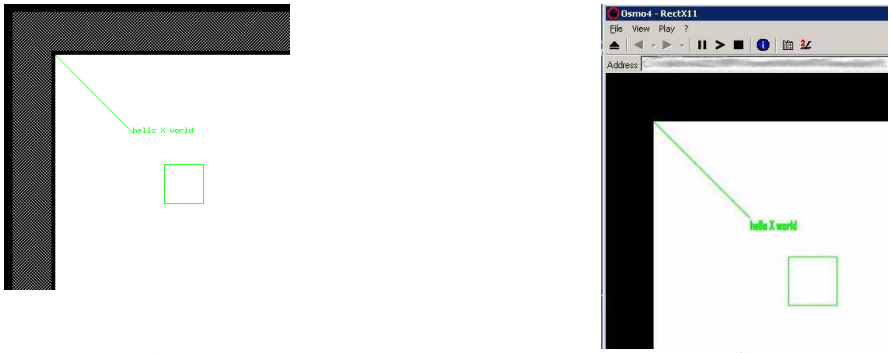
5. EXPERIMENTAL RESULTS

Graphical content evaluation

In order to check the effectiveness of the proposed architecture, three types of applications have been considered:

- static scenes composed by basic text and graphical content (see Figure 3.a);
- static scenes combining graphics and images, as generated by the GEDIT text editor (see Figure 4.a);
- animated graphical scenes as generated by the XCLOCK (see Figure 5.a).

It is to be understood that each time the graphical content is converted; this is without loss in visual quality, see Figures 3.b, 4.b and 5.b.



(a) (b)

Figure 3. Conversion of an “Hello world” scene

(a) (b)

Figure 4. Conversion of the content generated by the GEDIT

(a) (b)

Figure 5. Conversion of the content generated by the XCLOCK

Compression efficiency

The experiments focused on the amount of data required in order to represent animated graphics (the XClock), see Tables 1 and 2. Table 1 illustrates the amount of KBytes necessary to represent the XCLOCK in BiFS, as a function of time (the number of minutes for which the clock was running). It can be noticed that the largest amount of data per time unit corresponds to the initialization phase (t = 0 min), the rest of the behavior being quite scalable.

Table 1: The data size required for the BiFS graphical content representation.

Time [min]	0	1	2	3	4	5	10
Data size [KB]	6.9	26	45	69	90	111	325

We emphasize that the values in Table 1 correspond to "raw data", *i.e.* data containing only the BiFS graphics and no header information. Table 2 presents a comparison between the traffic generated by the same XClock application in two cases: (1) when the X graphics is first converted into MPEG and then streamed with the BiFS streamer and (2) when considering the "classical" framework, *i.e.* when the X graphics is sent through the VNC protocol. The last row in the Table 2 represents the compression gain (expressed as a ratio of the traffic sizes) obtained by using the MobiThin architecture.

Table 2: BiFS *vs*. VNC traffic.

Time [min]	0	1	2	3	4	5	10
BiFS [KB]	10	55	98	175	228	245	500
VNC [KB]	594	835	878	1100	1200	1290	1860
VNC/BiFS	59.4	15.18	8.9	6.28	5.2	5.2	3.72

Examination of results presented in the preceding table at first may lead us to conclude that improved performance decreases with time when, in fact, periodic screen reset operations will influence a tendency towards results at the start of the period.

User events

In Figure 6.a, "open the chocolate" is a text with a *TouchSensor* attached to it. When clicking on this text, a specific AJAX HTTP Request is made. This request is interpreted by the *Event handler* and the scene is updated after receiving the corresponding AJAX HTTP Response (the chocolate is opened), Figure 6.b. The response time is network dependent; in our environment it was less than 1ms.

(a) (b)

Figure 6."Click" handling

6. CONCLUSION

This paper intends to offer a PoC for the BiFS suitability in remote display graphical application situations. All experiments were successful: user expectation with respect to content quality and interactivity were met while the traffic was significantly lower (see Table 2). These experiments considered static, animated graphics and text alone. Some MobiThin applications (*e.g.* YouTube-like www browsing) will combine these types of content with multimedia content (video/audio). In such cases, the video/audio is directly attached to the BiFS scene and streamed in its native format (without any further conversion). In other words, the X11 content would contain a URL to the audio/video content to be subsequently streamed and played on the player. As a result, the MPEG advantages will be even more important in such real world situations then reported in Tables 1 and 2.

In order to have the complete architecture "fully connected", our next steps are focused on developing tree parts: (1) *Live BiFS streaming component* – streaming the input from the XParser, (2) *Extending the conversion* - extending the list of converted graphical primitives for running more complex application than xclock (*e.g.* browser, office) and (3) *Wireless environment* - by solving the streaming issue we could set-up the complete MobiThin environment by using a MPEG-4 player on a thin device(*e.g.* Pocket PC).

The current BiFS architecture still leaves several unaddressed issues relating to emerging remote display functionalities:

- direct collaborative work on the same document (be it of office or multimedia type) from different thin clients;
- the sharing of multimedia content (*e.g.* thin client video camera) generated by a particular user amongst a (social) network of thin clients;
- the possibility of a continuous work flow even when the network connection is lost;
- distributed and collaborative black or white boarding applications.

All these issues may be considered as potential MPEG-4 BiFS extensions[11].

ACKNOWLEDGEMENT

This work is supported by the FP7 MobiThin project (www.mobithin.eu).

REFERENCES

1. http://www.mobithin.eu
2. T. Richardson, Q. Stafford-Fraser, K. R. Wood, A. Hopper, “Virtual Network Computing”, IEEE Internet Computing, 1998.
3. Citrix Independent Computing Architecture, http://www.citrix.com
4. Microsoft Remote Desktop Protocol, http://support.microsoft.com/kb/186607
5. ISO/IEC JTC1/SC29/WG11 14496-11
6. http://www.chiariglione.org/mpeg/tutorials/papers/icj-mpeg4-si/05-BIFS_paper/5-BIFS_paper.htm
7. ISO/IEC JTC1/SC29/WG11 14496-20
8. http://www.w3schools.com/Ajax/Default.Asp
9. http://gpac.sourceforge.net/index.php
10. www.live555.com
11. B. Joveski, I.J. Marshall, M. Mitrea, F. Preteux, P. Simoens, “Active and reactive components for participative, distributed and collaborative BiFS scenes”, ISO/IEC JTC 1/SC 29/WG 11 M 16676, London, July 2009